Computer-Aided Construction of Ecological Simulation Models

COMPUTER-AIDED CONSTRUCTION OF ECOLOGICAL SIMULATION PROGRAMS

Robert Muetzelfeldt
Dave Robertson
Mike Uschold
Alan Bundy

DAI RESEARCH PAPER NO. 314

Paper submitted to IMACS - International Symposium on AI, Expert Systems and Languages in Modelling and Simulation. Barcelona, 1987.

Computer–Aided Construction of Ecological Simulation Programs

Robert Muetzelfeldt ‡, Dave Robertson †, Mike Uschold †, Alan Bundy †

†Department of Artificial Intelligence, University of Edinburgh.
‡Department of Forestry and Natural Resources, University of Edinburgh.

Abstract

We describe the development of ECO, a program which enables ecologists with minimal mathematical or computing skills to build simulation models. Two important problems are found in this system: the specification formalism (System Dynamics) is not sufficiently expressive; and the system requires that users adapt to its specification language, rather than using their own terminology. We describe solutions to both these problems. The Submodels system permits users to construct models using more complex computational structures than those found in System Dynamics. Use of a "high level" specification language, based on a typed logic, is proposed as a means of allowing the users to express their modelling requirements using ecological terminology. This approach is compact, expressive and provides a framework for guiding the user during the process of model construction.

1 Introduction

Many ecologists would like to construct computer simulations of ecological systems, but are discouraged by the complexity of current simulation languages. They must learn how to program in an available language (a considerable effort); or describe their model to a sympathetic modelling expert who will then implement their model on their behalf. Our goal is to relieve the ecologist of this handicap by providing a computer system which is easy to use and which helps the ecologist 'transform' their view of the problem from ecological terms, ultimately to a working computer program. This paper is an overview of our work over the last 3 years, describing our original system, its shortcomings, and our current and proposed solutions to these problems.

We begin, in Section 2, by describing our initial approach (based on a System Dynamics formalism) and stating why it is insufficient for our ultimate objective. We show that the main sources of its inadequacies are insufficient representative power of the chosen formalism; failure to bridge the terminology gap between user and machine; and failure to provide guidance about how to construct appropriate models. In Section 3 we describe a second working system which was developed to tackle the problems of representation by using a different formalism and interface

mechanism. Both these approaches lack expressiveness and also force users to work at the level of the modelling elements provided by each formalism. In Section 4 we explore the use of a typed logic for capturing expressive, high level statements about the system being modelled, allowing details of model structure to be inferred rather than provided directly by users.

2 ECO: An early prototype

2.1 Overview of the System

The original system, named ECO, (Uschold *et.al.* 1984) relies upon a System Dynamics formalism to express model structure. Entities in the real world are represented in the model by "tanks" with "flows" of some substance between them. Networks of equations may be used to regulate the rate of flow between tanks. This formalism can be manipulated by users, via an interface package, to produce a Task Specification (TS) for the model they require. A knowledge base of ecological relationships is used by the system to perform some simple checks for ecological consistency in the developing TS. When complete (as determined by the syntactical structure of the formalism), the TS is automatically converted into a target language (*e.g.* Fortran) and the simulation may then be executed. Recently, we have added the ability to run simulations directly in Prolog, our chosen implementation language, using a special purpose interpreter. This bypasses the code generation phase but does not effect the core of our research – the interface between user and formal TS.

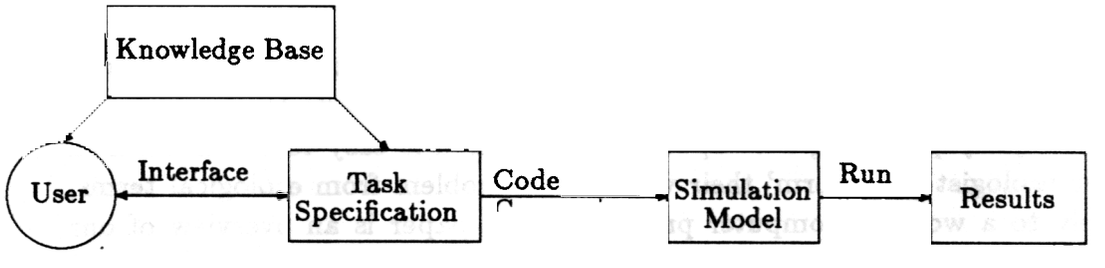


Figure 1: The original ECO system

Figure 1 shows the general layout of ECO. We shall describe several types of interface which have been used in ECO. We shall then explain why the System Dynamics formalism is inadequate for all but a restricted range of ecological models.

2.2 Interfaces

Given the general architecture and formalism described in Section 2.1, we address the problem of extracting the specification for a particular model from users. We experimented with 3 types of interface, which are described below.

Command–Driven Interface

Our first interface for ECO allowed users to input ecological statements in pseudo–English. The system parses these sentences and converts them into fragments of System Dynamics structure. For example, the user might input the sentence "sheep graze grass", which the system interprets as an instruction to create a compartment called sheep and a compartment called grass with a flow representing grazing between them. Since the system recognises the grazing relationship it is able to automatically orient the direction of flow from grass to sheep. Introducing these new structures creates unfilled "slots" which must be further specified. In this example the flow of grazing must be governed by a regulatory system of equations (*e.g.* the user might say "grazing uses grazing_equation_3") and the sheep and grass state variables must be set initial values (*e.g.* the user might say "assign sheep 10" and "assign grass 20"). Facilities for checking the syntactic completeness of the model and for summarising the model structure in text form are provided.

Unfortunately, there are severe problems with this command driven approach. First, it is entirely up to the user to describe what to do – no prompting of any sort is provided by the system. Second, the user must learn the syntax of the different sentence types and the potentially large number of ecological keywords recognised by the system. This is made more difficult by the apparent naturalness of some of the sentence forms, tempting users into thinking that they can type in any English sentence.

Form–Filling Interface

In response to the above problems, we have experimented with a form–filling approach. The task of this interface is to allow users to locate templates for particular commands and, having located the required template, to fill in the correct argument slots. An example of a typical display using this interface is shown in Figure 2. In this diagram, the user has chosen a "Process" command by picking the rectangle of that name using the mouse. In response, the system has displayed a "form" for this type of command which contains three slots for its arguments (the two interacting model components with a process between them). The user has filled these slots with the names which he/she wants to use in the

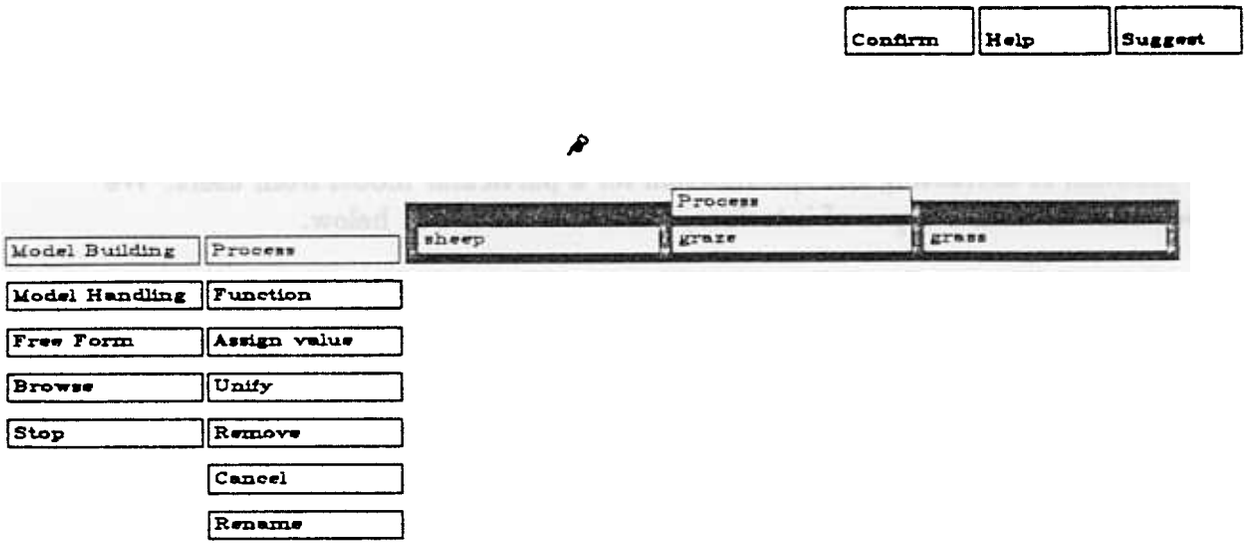


Figure 2: A form filling interface.

command and can now send the completed form for incorporation into the TS by picking the "Confirm" button. Alternatively, the user could obtain help about the meaning of any of the commands or get suggestions for the names which could fit into a given command slot.

An advantage of this approach is that users are not given the impression that the system can understand arbitrarily phrased commands because the structure of each command is displayed explicitly. There is also no need to learn the syntax of each command because the system displays the appropriate keywords and provides guidance in selecting the appropriate arguments for a given command. However, two major problems remain. First, the user must still decide what to do at each point, although he/she is helped by seeing a menu of possible actions. Second, a large number of pointing, clicking and typing operations are required to specify a model of even moderate complexity.

Graphics Interface

An alternative approach is to remove the need for a command language altogether and simply permit users to manipulate the constructs of the formalism directly. To do this, an icon is provided for each System Dynamics symbol (compartment, material flow, information flow, flow regulator, *etc*). A model is constructed by dragging icons to a position on the screen and locking them together. We have implemented an interface based on these principles and this strategy is adopted by other programs, such as Stella (Lewis 1986). This approach is attractive for

those modelling problems which fit readily into the compartment–flow framework. However, the user must be familiar with the symbolism and the resulting model diagram can be messy, even for quite simple models.

2.3 Critique of the System Dynamics based systems

In previous sections we have described three different interfaces to a System Dynamics TS formalism. In all of these interfaces it has been possible to utilise two inherent advantages of the formalism:

- **The flow analogy.** Ecologists frequently think of ecosystems in terms of compartments and flows. This makes System Dynamics a useful formalism for modelling such systems.
- **Simple building blocks.** The number of structure types to be manipulated is small and there is also a small number of syntactically permissible connections between structures. This means that the interface can utilise rigid syntactic restrictions between model structures to control model construction.

Unfortunately, the System Dynamics formalism is limiting in several ways:

- **It may be tedious.** A model may contain a number of compartments of the same type (*e.g.* several herbivore species). It is tediously repetitive to have to set up each compartment, with its associated flows, using the same set of instructions again and again.
- **It may be inappropriate.** Users must think in terms of the System Dynamics formalism rather than simply providing ecological information about their system. This creates problems when the flow analogy does not apply to a user's model. For example, an ecologist who wanted to model movement of an organism using Eco would have to create state variables representing the X and Y coordinates of the organism and set up "flows" of some abstract substance representing the rate of change of coordinates.
- **It may be inadequate.** Certain model structures can only be represented by extending the System Dynamics formalism, or cannot be represented at all. For example, a model of an animal population with age classes may require the continuous representation of mortality but the discrete (annual) shifting along of population numbers between age classes. More seriously, nobody would contemplate using System Dynamics to model a forest in which the number of trees and the branching structure of each tree change over time.

In short, the System Dynamics formalism is suitable for models representing flows of material between objects but becomes confusing or inadequate when more complex relationships are required. There is a tension between the need to add special purpose model structures to the specification language (thus extending the System Dynamics formalism) and ensuring that naive users can manipulate those structures easily (since each new structure adds to the potential complexity of the user interface). In Section 3 we describe a program which allows complex structures to be accommodated within the specification without forcing the user to understand the internal details of these program elements. This helps reduce the tension between complexity of the specification and simplicity of the interface.

3 The Submodels System

We originally developed the submodels modelling system in response to the widespread use of Fortran, BASIC and batch-orientated simulation languages for the implementation of ecological simulation models. This often results in unwieldy programs which are difficult to construct and modify; it makes it difficult to convey the structure of the model to others; it creates problems in trying to build up a model from pre-programmed submodels; and it inhibits attempts to explore the behaviour of a subset of the submodels without running the whole model. An environment for implementing ecological models which overcame these problems would be a valuable modelling tool.

At the same time, we hoped to develop a method for representing model structure which had the simplicity and conceptual appeal of the System Dynamics formalism, and yet which could cater for a much broader range of ecological models. Also, we required a formalism for representing model structure which could serve as a vehicle for testing mechanisms for intelligent guidance during the process of constructing a model.

3.1 The basis of the submodel modelling approach

As implied by its name, the Submodels formalism is based on the notion that the structure of a model can be defined by the collection of submodels it contains, and by the links between them. This may be though of as corresponding to the use of subroutines in a conventional Fortran solution to the problem of programming large simulation models. Alternatively, one can consider that the submodels in an ecological model correspond to ecological components that together make up some ecological system, such as a tree, a forest or a complete forest ecosystem.

The submodels may be of one of two types: base submodels and built submodels. A base submodel is a pre-programmed element representing a dynamic

modelling component (such as a deer population). It can be arbitrarily complex internally, but, as far as the model builder is concerned, it is characterised completely by a set of state variables, input parameters and output variables. A built submodel, on the other hand, consists of a number of submodels, which also can be base or built submodels. This approach is thus strongly hierarchical, and submodels can be nested to any depth, just as subroutines can be in a Fortran–based approach, or just as one can consider that ecological components themselves consist of subcomponents. Both base and built submodels can be stored in a submodel library, ready to be selected for inclusion in other models, and the user need not know whether a submodel is base or built when selecting it.

Communication between submodels can be of two forms. In the simplest case, an input of one submodel is unified with (equated to) the output of another. For example, a forest submodel might have an input *size of insect population*, which could be unified with the corresponding output variable from the insect population submodel. In Fortran terms, this corresponds to the use of the same variable name in the argument list for two subroutine calls.

The other form of communication between submodels involves the use of one or more functional relationships between the output of one submodel, and the input of another. For example, in the above case the forest submodel might have an input *rate of defoliation* rather than *size of insect population.* In this case *rate of defoliation* needs to be calculated as a function of the appropriate output from the insect population submodel.

There are two points to note about the use of functional relationships in this context. First, it is quite possible to have a network of functions between the output of one submodel and the input of the other. In this case, the user needs to define intermediate variables to permit the functions to be linked together, and these then constitute outputs of the submodel at the next highest level. Second, each introduced function will have a number of parameters, which then constitute inputs for the submodel at the next highest level.

3.2 Implementation of the submodels approach

We have implemented a graphics-based interface for building ecological models according to the above concepts. The user begins with an empty box which represents the "current model". Typically, the user would then search through the library of submodels to find and load one which is appropriate to the modelling objectives. This searching is currently performed by using the mouse on a Sun 3 workstation to move through a hierarchically-organised set of menus leading to individual submodels (base or built) in the library. We intend eventually to

incorporate a more powerful browsing mechanism (*e.g.* Robertson *et.al.* 1985). The selected submodel is then positioned on the screen (again using the mouse) within the "current model". Further submodels can then be added.

The user can unify submodel variables by clicking the mouse on the appropriate input and output variables. A functional relationship between variables can be incorporated in one of two ways. The user can load an appropriate mathematical function from the function library and then unify the corresponding inputs and outputs, using an identical approach as for the submodels. Alternatively, the user can specify that there is an influence (as yet unspecified) between the output of one submodel and the input of another, and put into abeyance the need to specify the particular mathematical form of the relationship: if necessary, he/she can be reminded by the program to provide this at some later time.

The user can move up and down through the hierarchically-organised model. On moving up, the submodel on which he/she has just been working appears as a small box, just one of the several submodels at that level. On moving down into a built submodel (either loaded as a library built model or built during the current session), the user can see and manipulate the structure of that submodel.

The completed model can be run, and the results of the simulation plotted. Currently, the simulation code is in Prolog, the language in which the other parts of the system are implemented. This simplifies the task of developing a prototype of this modelling system, and does not restrict the complexity of the base models, but does mean that the simulation runs slowly compared with Fortran.

3.3 Critique of the submodels modelling system

- **Model complexity.** The submodels modelling system bypasses the problem of allowing complex modelling procedures to be represented, by permitting the use of pre-programmed base submodels containing arbitrarily-complex code. Since these are in effect "black boxes", we are not concerned with representing the structure of the base submodel other than in procedural form, and we do not need to concern ourselves with displaying their structure to the user: we are only concerned with the interface between the base submodels and the rest of the model. This approach permits us to adapt very rapidly the many published models in the ecological literature, such as plant growth models, models of the population dynamics of particular species, etc. However, it does have an important drawback, that the submodel should be provided with an interface which anticipates all conceivable interactions with other submodels. For example, it may not be possible to use a submodel of forest growth to investigate insect defoliation if that submodel does not have a *rate of defoliation* input.

- **Conceptual appeal: components and influences.** The System Dynamics formalism is conceptually appealing only for that subset of all ecological models that can be visualised in terms of flows between tanks. We believe that a major strength of the submodels approach is that it maps very closely to the way that ecologists begin by describing an ecological system in terms of the ecological components the system contains, and the influences (cause-effect relationships) within the system. This allows us to use the following metaphor for the ecologist new to the program: that the blank screen corresponds to a blank piece of land, which can be populated with animals, plants and climatic components. Having done this, the interactions between components can be specified.

- **Conceptual appeal: hierarchical organisation.** The hierarchical structure of the submodels approach is very natural to ecologists, corresponding to the hierarchy of *community, population, organism* in natural systems.

The submodels approach provides much greater flexibility in modelling than an approach based on implementing models in a conventional programming or simulation language: model structure can be changed interactively. The approach corresponds to the way that many ecologists describe the system they wish to model, and it permits considerable complexity in the mathematical basis of a model to be embedded. However, this expressive power is to some extent illusory: the user cannot see inside the base models, he/she cannot manipulate their internal structure, and he/she cannot himself express arbitrarily-complex modelling ideas.

4 Building a Model by Selecting Ecological Statements

When an ecologist consults a modeller for help in building a model, he will utter a variety of statements which together form a concise specification for the required model. However, those statements will not appear in the final implementation: they will rarely refer directly to the equations or control structures that are used to represent the model in some programming language. Rather, the modeller will use the ecologists statements to **infer** details of model structure. Furthermore, the statements could very well lead the modeller to conclude that aspects of the model require novel, non–standard programming solutions.

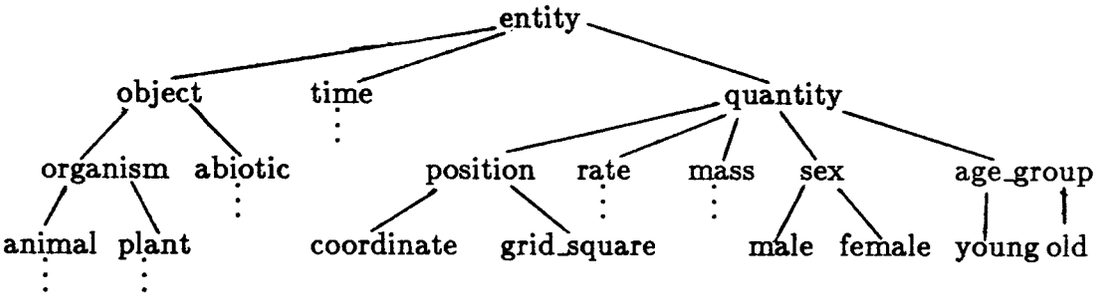
How can we hope to capture such statements in a form which may be used by the computer? The use of a logic language suggests itself, because of its proven expressive power and potential for inference. In order to proceed with this

approach, we must address a number of issues which are discussed in subsequent sections.

- How to formally represent high level ecological statements.
- How to extract these statements from the user.
- How to convert the high level TS into the low level TS.
- How to use the logic statements to suggest appropriate model structures.

4.1 Formalising High Level Ecological Statements

Ecologists often use simulation models to represent objects in the real world and define interactions between different types of object. The vocabulary used to describe these interactions is rich in statements conveying a special meaning in terms of model structure (*e.g.* "age class", "location"). Therefore, we require a representation language which has a formal syntax but allows flexibility in defining representations of statements. We have chosen a typed logic for this purpose because of its success as a general specification language (*e.g.* Balzer and Goldman 1979, Burstall and Goguen 1981) and because we have been able to use it to represent formal statements of ecological theory from Niven 1982 and Smith 1974, as well as application models (*e.g.* Hilborn and Sinclair 1979). We shall not dwell on the details of the formalism in this paper and refer the reader to McSkimin and Minker (1979) for a thorough description. The benefit of typed logic for our purpose is that it allows definitions of functions which refer to object types. These types may be organised into a hierarchy with nodes representing object types and arcs representing subtype relations, an example of which is shown below.



Definitions in the logic of sentences involving these objects can seem confusing to the uninitiated. For example, the English statement "All young, male wolves have a biomass of 25" is written in the logic as:

$$\forall X \in wolf \;\; \forall T \in time$$
$$age\_class(X,T) = young \;\&\; sex(X) = male \rightarrow biomass(X,T) = 25$$

To simplify this discussion, we shall provide only the English description of logic sentences, in the knowledge that they can easily be converted into the logic (see Robertson *et.al.* 1986 for a detailed description). Our chosen implementation language (Prolog) supports the incorporation of logic based statements into the program.

This formalism can represent the same sorts of statements which were available in System Dynamics. For example, the user can provide process definitions – such as "sheep graze grass" or "sheep respire" – or talk about rates of interaction – for example "the rate of respiration is 10". However, there are several important advantages of the logic representation over System Dynamics (or similar) formalisms:

- **Economy.** A single statement in the logic will frequently unpack into many statements in the target language for implementing the model. For example, in a model containing several objects of type animal and several objects of type plant a user might say "All animals eat all plants". Given that plants and animals are organisms (from the type hierarchy), one could also say "All organisms respire" which means that any object of type plant or animal will respire.

- **Expressiveness.** It can represent concepts which are beyond the capabilities of System Dynamics. For example, a user can say "If any predator is in proximity to any prey, the prey will be eaten". This statement uses a test for proximity (which can also be defined in the logic) as a condition for an interaction between two types of object.

- **Abstract representation.** The statements represented in the model do not have to directly correspond to single structures in the final model. For instance, a user might say "All animals have some location" without specifying at that time how the location of animals should be modelled (*e.g.* coordinates,zones,etc).

The practical value of this method is that it permits users to input what, to them, look like familiar ecological statements but which the computer can process using the syntax of the underlying logic. In Section 4.3 we shall describe how high level statements may be used to derive a quantitative simulation model. First, we propose a user interface to the high level specification.

4.2 Interface to the High Level Specification

A large collection of possible logic sentences representing ecological statements will be contained in the system knowledge base. The types referred to in these sentences will be at their most general (*e.g.* "All animals are predators" is the most general predator statement which a user could make in a particular model). Thus, the statements in the knowledge base may be thought of as constituting a pool of templates from which the user must select those applicable to his/her model and restrict to the appropriate types of object. Using a high level specification poses three problems:

1. Since the high level statements represent information about an ill defined ecological domain, there is no way of ensuring that all possible ecological statements are capable of being handled by the system. To avoid the confusion which we saw in Section 2.2 between what users expect the system to be capable of representing and what it actually can represent, users must be able to examine the knowledge base to see if the system is able to represent their ideas.

2. The logic sentences represent ecological statements but the logical notation would be unfamiliar to most ecologists. Therefore, a mechanism for displaying sentences in an acceptable format is required.

3. Users must be able to restrict the types of object referred to by selected sentences and, in some cases, to construct new complex sentences from simpler templates. The mechanism for doing this job must be simple to use.

Rudimentary programs for solving problems 1 and 2 have already been implemented but require further refinement. We have yet to implement a program for solving problem 3 but provide a sketch of our proposed solution. For a more thorough description see Robertson *et.al.* (1986).

Users can gain access to sentence templates by moving between items in the type hierarchy using a browsing mechanism (Robertson *et.al.* 1985 describes one possible browsing mechanism) and selecting types which they want to consider. They are then given access to all sentences which refer to those types (*e.g.* if a user wanted to talk about *age_class* he/she would select that node in the hierarchy). From a display of candidate sentences, users can select those sentences which apply to their model and restrict their types, using a menu of editing commands. When satisfied that the currently selected templates match his/her required description of the model, the user may insert them into the high level TS.

Joining templates to form more complex sentences requires the facility to point out the appropriate templates from those which are currently selected; place them in the correct order; and connect them with the appropriate logic connectives (*e.g.* conjunction, disjunction, implication). We hope to implement this using a bit–map display, displaying templates on detectable panels which can be moved around using a mouse. In this way users who required sentences which the system did not provide directly could assemble them by "cutting" out the constituent templates and "pasting" them into new sentences. The following example shows why this is necessary.

Suppose the system provides the following separate sentences:
1 – "All organisms have an age class at all times".
2 – "All organisms have a sex at all times".
3 – "All organisms have a biomass at all times".

The user may want to use these as building blocks to state:
"All young male organisms have biomass 25."
This can be done by joining sentences 1 and 2; using these to imply sentence 3; and constraining the types of age class, sex and biomass to be young, male and 25, respectively.

4.3 Deriving the Low Level Specification

Assuming that some logic statements have been inserted into the high level TS by the user, how can these be used to derive a low level computational structure for the model ? In Section 4.1 we drew attention to the fact that many ecological statements had specific meanings in terms of model structure. For example, if the high level TS contains the sentence "All animals have a coordinate location" then the low level TS must contain a structure representing the X and Y coordinates of each organism of type animal. Certain parameters of this structural schema might need to be further specified by the user (*e.g.* the maximum and minimum values of the X and Y coordinates). Therefore, the high level TS may be thought of as constituting information which allows the computer to isolate appropriate low level model structures, which may be quite complex, using formal ecological statements, which are comparatively simple. If these derived low level schemata can be represented in a form similar to that used in the "Submodels" system (Section 3), then we shall have a system which can represent a model in users' terminology while retaining the low level representational power required for non–trivial models. Further research is required to ensure that this happy marriage is consummated but, even if this takes place, there remains a need for intelligent guidance during the modelling process. We next discuss one possible solution to

this problem using a typed logic approach.

4.4 Generating Suggestions

The logic specification system, as described above, leaves the onus on the user to decide which statements to include in his/her high level specification. However, a simple additional mechanism may be added to allow inference of possible new model structures from the set of statements comprising the high level TS. To explain how this mechanism works, consider the following simplified example.

Assume that the pool of possible ecological statements in the knowledge base contains the following sentence templates:

1 – "All organisms are represented as individuals."
2 – "All organisms move".
3 – "If organisms are represented as individuals and those organisms move then those organisms have X and Y coordinates"

Also assume that the user's TS contains the following sentences, obtained by selecting templates 1 and 2 and restricting the types to which they refer:

A – "All sheep are represented as individuals."
B – "All sheep move".

The system can recognise that the conditions for template rule 3 are satisfied by A and B above, given the substitution of sheep for organisms (*i.e.* "sheep are represented as individuals and those sheep move"). It can therefore infer the fact that "sheep have X and Y coordinates" and suggest to the user that this hitherto undiscovered statement should be included in the TS.

This simple mechanism permits the computer to store and utilise ecological rules as a means of providing suggestions to the user, given a partially constructed model. It therefore represents a first step towards a modelling system which is capable of actively helping users decide upon a correct model structure, rather than passively responding to users' decisions. We hope in future to elaborate upon this basic mechanism – for example, by distinguishing general modelling strategies which relate to users' goals for their completed models. For a discussion of these extensions, see Uschold *et.al.* 1986.

5 Conclusion

We have described a working system (ECO) which enables ecologists who may possess minimal mathematical or computing skills to build simulation models. This system was useful for users who were familiar with the TS formalism but

useless for those who were not able to translate their mental image of the model (in ecological terms) into the language used by the system. There are also a significant number of ecological models which simply cannot be expressed in System Dynamics notation. We have attacked this problem on two fronts: increasing the representational power of the specification language by permitting networks of "submodels" containing complex code; and designing a method for formally representing modelling statements. in ecological terminology. Since these ecological statements can be used to isolate computational structures appropriate for a user's model, it seems possible that the two advantages could be combined in one system which naive ecologists would find easy to use while being capable of building simulation models of reasonable complexity. While these enchancements would in themselves be useful to ecological modellers, there remains a need for the computer to help users make strategic decisions about what to specify in their models, taking into account the users objectives for the completed model.

Acknowledgements

This work was funded by SERC grant GR/E/00730. We are grateful to members of the Mathematical Reasoning Group in the Department of Artificial Intelligence at Edinburgh University for their practical advice and support during the course of this project.

References

Balzer, R., and Goldman, N. 1979, *Principles of Good Software Specification and their Implications for Specification Languages*, IEEE Computer Society, 58–67.

Bundy, A., 1984, *Intelligent Front Ends*, in Pergammon Infotech State of the Art Report on Expert Systems, (available from Dept. Artificial Intelligence, Edinburgh University as Research Paper 227)

Burstall, M. and Goguen, J., 1981, *An Informal Introduction to Specifications Using Clear*, Academic Press.

Hilborn, R. and Sinclair, A.R.E., 1979, *A Simulation of the Wildebeest Population, Other Ungulates and Their Predators*, in Sinclair and Norton–Griffiths (eds) *Serengetti: Dynamics of an Ecosystem*, University of Chicago Press.

Lewis, J., 1986, *STELLA: A Model of its Kind*, Practical Computing, September, 66–67.

McSkimin, J.R. and Minker, J., 1979, *A Predicate Calculus Based Semantic Network for Deductive Searching*, in Findler, N.V. (ed), *Associative Networks: Representation and use of Knowledge by Computers*, Academic Press.

Niven, B.S., 1982, *Formalisation of the Basic Concepts of Animal Ecology*, Erktennis, 17, 307–320.

Robertson, D., Muetzelfeldt, R., Plummer, D., Uschold, M. and Bundy, A., 1985, *The Eco Browser*, in Merry, M, (ed), *Expert Systems 85*, British Computer Society Specialist Group on Expert Systems, Cambridge University Press.

Robertson, D., Bundy, A., Uschold, M. and Muetzelfeldt, R., 1986, *Synthesis of Simulation Models from High Level Specifications*, submitted to IJCAI–87, available from the authors.

Smith, J.M., 1974, *Models in Ecology*, Cambridge University Press.

Uschold, M., Harding, N., Muetzelfeldt, R. and Bundy, A., 1984, *An Intelligent Front End for Ecological Modelling*, in O'Shea, T. (ed), Advances in Artificial Intelligence, also in Proceedings ECAI–84 available from Dept. Artificial Intelligence, Edinburgh University as Research Paper 223)

Uschold, M., 1986, *Computer–Aided Design of Program Specifications in the domain of Ecological Modelling*, Discussion Paper 35, Dept. Artificial Intelligence, Edinburgh University.